Python Basics

Python Interpreter

Start Python

Windows

```
C:\> python
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Mac or Linux

```
$ python3
Python 3.5.2 (default, Aug 18 2017, 17:48:00)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Use Interactive Interpreter as a Calculator

```
$ python3
Python 3.5.2 (default, Aug 18 2017, 17:48:00)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 10-4
6
>>> 2*4
8
>>> 20/5
4
>>> 3**2
9
```

Mathematical expressions are evaluated:

- **P**arentheses
- **E**xponents
- **M**ultiplication and
- **D**ivision
- **A**ddition and
- **S**ubtraction

Use Interpreter to print Hello World

- Strings can be enclosed with single quotes or double quotes.
- To remove the single quotes in the output, use the print command.

```
>>> "Hello World!"
'Hello World!'
>>> 'Hello World!'
'Hello World!'
>>> print("Hello World!")
Hello World!
```

Quit the Interpreter and Start IDLE

- Python includes the Integrated Development Environment (IDLE)
- Windows - open IDLE from the Start menu
- Mac or Linux - open IDLE from the command line.

Windows

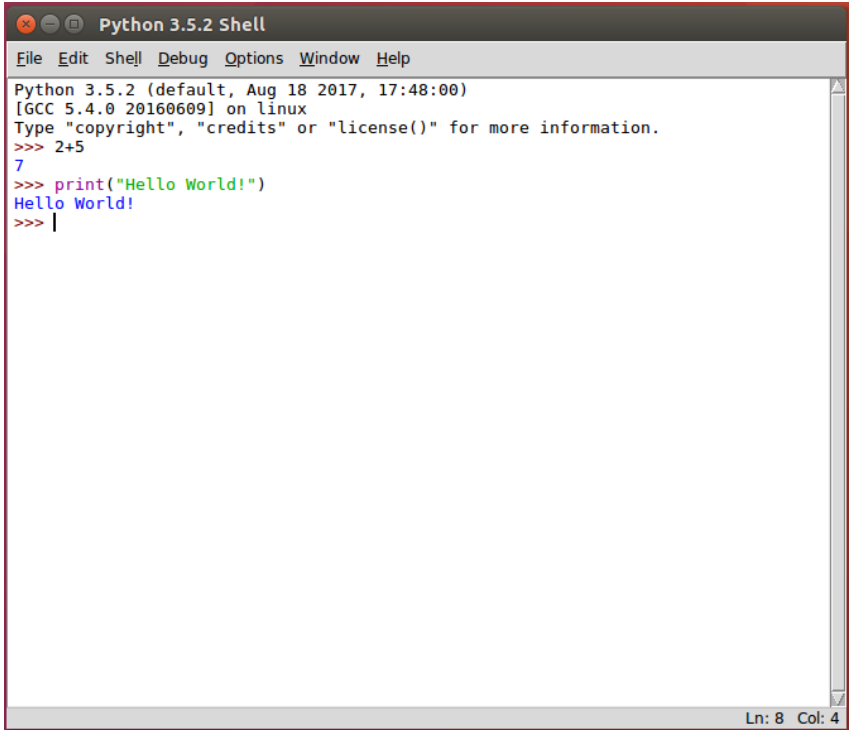
Start > Python 3.6 > IDLE (Python 3.6 32-bit).

Mac or Linux

```
>>> "Hello World!"
'Hello World!'
>>> 'Hello World!'
'Hello World!'
>>> quit()
$ idle3
```

IDLE Benefits

- Provides color coding
- Includes a text editor for writing programs
- Quickly save and run programs



Activity - Write, Save, and Run Your First Program

1. In IDLE, click **File > New File (Ctrl+N)** to open an Untitled script file.
2. Save the file as 01_hello-world.py in your GitHub project directory.
3. Enter the following in the script:

```
print("Hello World!")
```

4. Save the script; click **File > Save (Ctrl+S)**
5. Run the script; click **Run > Run Module (F5)**

First Program and Output

01_Hello-World.py - /home/allan/Dropbox/ETW-BP/01_Hello-W

File Edit Format Run Options Window Help

```
print("Hello World!")
```

Program File

Ln: 2 Col: 0

Python 3.5.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.5.2 (default, Aug 18 2017, 17:48:00)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+5
7
>>> print("Hello World!")
Hello World!
>>>
=========== RESTART: /home/allan/Dropbox/ETW-BP/01_Hello-World.py
===========
Hello World!
>>>
```

Output

Ln: 11 Col: 4

Data Types, Variables, and Conversions

Basic Data Types

- The four basic data types we will use are:
  - Integer
  - Float
  - String
  - Boolean
- Use the type() command to determine the data type.

```
print(type(98))    #<class 'int'>
print(type(98.6)) #<class 'float'>
print(type("Hi!"))#<class 'str'>
print(type(True)) #<class 'bool'>
```

Boolean Comparison Operators

Operator	Meaning
>	Greater than
<	Less than
==	Equal to
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

```
>>> 1<2
True
>>> 1>2
False
>>> 1==1
True
>>> 1!=1
False
>>> 1>=1
True
>>> 1<=1
True
```

Creating and Using a Variable

- Use a single equal sign to assign a value to a variable.
- A variable can then be called for other operations.

```
>>> x=3
>>> x*5
15
>>> "Cisco"*x
'CiscoCiscoCisco'
```

Concatenate Multiple String Variables

- Concatenation is the process of combining multiple strings.

```
>>> str1="Cisco"
>>> str2="Networking"
>>> str3="Academy"
>>> space=" "
>>> print(str1+space+str2+space+str3)
Cisco Networking Academy
>>>
```

Converting Data Types

- Concatenation does not work for different data types.

```
>>> x=3
>>> print("This value of X is " + x)
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    print("This value of X is " + x)
TypeError: Can't convert 'int' object to str
implicitly
```

Converting Data Types

- Use the **str()** command to convert the data type to a string.

```
>>> x=3
>>> print("The value of x is " + x)
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    print("This value of X is " + x)
TypeError: Can't convert 'int' object to str
implicitly
>>> print("The value of x is " + str(x))
The value of x is 3
>>>
```

Converting Data Types

- The type for the variable x is still an integer.

```
>>> x=3
>>> print("The value of x is " + x)
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    print("This value of X is " + x)
TypeError: Can't convert 'int' object to str
implicitly
>>> print("The value of x is " + str(x))
The value of x is 3
>>> type(x)
<class 'int'>
```

```
string1="33"
valor=int(string1)+5
print(valor)
```

Converting Data Types

- To convert the data type, reassign the variable to the new data type.

```
>>> x=3
>>> print("The value of x is " + x)
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in
<module>
    print("This value of X is " + x)
TypeError: Can't convert 'int' object
to str implicitly
>>> print("The value of x is " +
str(x))
The value of x is 3
>>> type(x)
<class 'int'>
>>> x=str(x)
>>> type(x)
<class 'str'>
```

Converting Data Types

- Use **“{:.2f}”.format** to display a float to two decimal places.
- Change the **2** to increase or decrease decimal places.

```
>>> pi = 22/7
>>> print(pi)
3.142857142857143
>>> print("{:.2f}".format(pi))
3.14
>>>
```

Actividad

- Referente al lenguaje Python, Investiga y documenta cómo:
- Cómo redondear y truncar números con n valores en enteros o decimales.
- El valor es asignado por el usuario.
- Ejemplo:

Ejemplo:

Indica un valor numérico: 99.6794936

El valor redondeado a enteros es: 100

El valor redondeado a enteros con 1 decimal es: 99.7

El valor redondeado a enteros con 2 decimales es: 99.68

El valor truncado a enteros es: 99

El valor truncado a entero con 1 decimal es: 99.6

El valor truncado a entero con 2 decimales es: 99.67

Lists and Dictionaries

Lists

- A list is an ordered list of items.
  - Create a list using the brackets **[ ]** and enclosing each item in the list with quotes.
  - Use the **type()** command to verify the data type.
  - Use the **len()** command return the number of items in a list.
  - Call the list variable name to display it's contents.

```
>>> hostnames=["R1","R2","R3","S1","S2"]
>>> type(hostnames)
<class 'list'>
>>> len(hostnames)
5
>>> hostnames
['R1', 'R2', 'R3', 'S1', 'S2']
```

Listas

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

```
print(lista[0]) # 1
print(lista[1]) # 2.5
print(lista[2]) # cadena
print(lista[3]) # [5,6]
print(lista[3][0]) # 5
print(lista[3][1]) # 6
print(lista[1:3]) # [2.5, 'cadena']
print(lista[1:6]) # [2.5, 'cadena', [5, 6], 4]
```

Una lista es una estructura de datos y un tipo de dato en python con características especiales. Permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras listas.

Listas

Puedes recorrer una lista utilizando un for.

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

```
for element in lista:
        print(element)
```

Métodos de Listas

Append()

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

Este método nos permite agregar nuevos elementos a una lista.

```
lista.append(10) # [1, 2.5, 'cadena', [5,6] , 4, 10]

lista.append([2,5]) [1, 2.5, 'cadena', [5,6] , 4, 10, [2,5]]
```

Una lista dentro de otra lista representa un solo elemento.

Métodos de Listas

Extend()

Permite agregar elementos dentro de una lista.
Al agregar una lista, cada elemento de ésta nueva lista se agrega como un elemento individual.

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

```
lista.extend([88,99]) # [1, 2.5, 'cadena', [5,6] , 4, 88, 99]
```

```
lista=["a","b","c","d","e"]
lista.insert(2,"Z")
print(lista)
```

Métodos de Listas

Remove()

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

El método remove quita de la lista el elemento (valor) que se le indique como parámetro.

```
lista.remove('cadena') # [1, 2.5, [5,6], 4]
```

Métodos de Listas

Index()

```
lista = [1, 2.5, 'cadena', [5,6] ,4]
```

Index devuelve el índice del primer elemento que le pasemos como parámetro.

```
lista.index('cadena') # 2
```

Métodos de Listas

Count()

```
lista = [7, 2.5, 'cadena', 7,[5,6] ,7]
```

Muestra cuántas veces se repite un elemento dentro de la lista.

```
lista.count(7) # 3
```

Métodos de Listas

Copy()

```
lista = [7, 2.5, 'cadena', 7,[5,6] ,7]
```

Copia el contenido de una lista en otra.

```
Otra_lista = lista.copy()
```

```
print(otra_lista) # [7, 2.5, 'cadena', 7,[5,6] ,7]
```

Métodos de Listas

Reverse()

```
lista = [7, 2.5, 'cadena', 7,[5,6] ,7]
```

Invierte el orden de los elementos de una lista. No devuelve valores.

```
lista.reverse() # [7, [5, 6], 7, 'cadena', 2.5, 7]
```

```
for element in lista:
        print(element)
```

Métodos de Listas

Sort()

```
lista = [7,40,10,3,5,60,32]
```

Ordena ascendentemente a > z .
No retorna valores.

```
lista.sort() # [3, 5, 7, 10, 32, 40, 60]

lista.sort(reverse=True) # [60, 40, 32, 10, 7, 5, 3]
```

Métodos de Listas

max(lista) y min(lista)

```
lista = [7,40,10,3,5,60,32]
```

Obtiene los valores máximos y mínimos de una lista

```
max(lista) # 60

min(lista) # 3
```

Lists

- Use the index to refer to an item and manipulate the list
  - The first item in a list is indexed as zero, the second is indexed as one, and so on.
  - The last item can be referenced with index **[-1]**
  - Replace an item by assigning a new value to the index.
  - Use the **del list[index]** command to remove an item from a list.

```
>>> hostnames=["R1","R2","R3","S1","S2"]
>>> type(hostnames)
<class 'list'>
>>> len(hostnames)
5
>>> hostnames
['R1', 'R2', 'R3', 'S1', 'S2']
>>> hostnames[0]
'R1'
>>> hostnames[-1]
'S2'
>>> hostnames[0]="RTR1"
>>> hostnames
['RTR1', 'R2', 'R3', 'S1', 'S2']
>>> del hostnames[3]
>>> hostnames
['RTR1', 'R2', 'R3', 'S2']
>>>
```

Dictionaries

- A list of unordered key/value pairs
  - Create a dictionary using the braces **{ }**
  - Each dictionary entry includes a key and a value.
  - Separate key and values with a colon.
  - Use quotes for keys and values that are strings.

```
>>> ipAddress =
{"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}
>>> type(ipAddress)
<class 'dict'>
```

Dictionaries

- Use the key to refer to an entry
  - The key is enclosed with brackets **[ ]**.
  - Keys that are strings can be referenced using single or double quotes.
  - Add a key/value pair by setting the new key equal to a value.
  - Use ***key*** **in** ***dictionary*** command to verify if a key exist in the dictionary

```
>>> ipAddress =
{"R1":"10.1.1.1","R2":"10.2.2.1","R3":"
10.3.3.1"}
>>> type(ipAddress)
<class 'dict'>
>>> print(ipAddress)
{'R1': '10.1.1.1', 'R2': '10.2.2.1',
'R3': '10.3.3.1'}
>>> ipAddress['R1']
'10.1.1.1'
>>> ipAddress["S1"]="10.1.1.10"
>>> ipAddress
{'R1': '10.1.1.1', 'R2': '10.2.2.1',
'R3': '10.3.3.1', 'S1': '10.1.1.10'}
>>> "R3" in ipAddress
True
>>>
```

Dictionaries

```
>>> ipAddress = {"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}
>>> type(ipAddress)
<class 'dict'>
>>> print(ipAddress)
{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1'}
>>> print(ipAddress['R1'])
'10.1.1.1'
>>> ipAddress["S1"]="10.1.1.10"
>>> print(ipAddress)
{'R1': '10.1.1.1', 'R2': '10.2.2.1', 'R3': '10.3.3.1', 'S1': '10.1.1.10'}
>>> print("R3" in ipAddress)
True
>>>
```

Dictionaries

- Values in a key/value pair can be any other data type including lists and dictionaries.

```
>>> ipAddress = {"R1":"10.1.1.1","R2":"10.2.2.1","R3":"10.3.3.1"}
>>> ipAddress["R3"] = ["10.3.3.1","10.3.3.2","10.3.3.3"]
>>> print(ipAddress)
{'S1':'10.1.1.10','R2':'10.2.2.1','R1':'10.1.1.1','R3':['10.3.3.1',
'10.3.3.2', '10.3.3.3' ] }
>>>
```

Diccionarios

```
for akey in diccionario:
        print (akey, ":", diccionario[akey] )
```

Estructura de datos.
Identifica a cada elemento por una clave (key).
El valor puede ser cualquier tipo de dato.

```
diccionario = {
        'nombre' : 'Carlos',
        'edad' : 22,
        'cursos': ['Python','java','JavaScript']
}
```

```
print (diccionario['nombre']) #Carlos
print (diccionario['edad'])#22
print (diccionario['cursos'])#['Python','java','JavaScript']
print (diccionario['cursos'][0]) # Python
```

Métodos de Diccionarios

keys()

Retorna las claves del diccionario.

```
diccionario = {'a':1, 'b':2, 'c':3, 'd':4}
keys= diccionario.keys()
print(keys)  # dict_keys(['a', 'b', 'c', 'd'])
```

Métodos de Diccionarios

values()

Retorna los valores del diccionario.

```
diccionario = {'a':1, 'b':2, 'c':3, 'd':4}
valores = diccionario.values()
print(valores)  # dict_values([1, 2, 3, 4])
```

Métodos de Diccionarios

clear()

Elimina los elementos (claves y valores).

```
diccionario = {'a':1, 'b':2, 'c':3, 'd':4}
diccionario.clear()
print(diccionario)  # {}
```

Métodos de Diccionarios

get()

Devuelve el valor de la clave.

```
diccionario = {'a':1, 'b':2, 'c':3, 'd':4}
print(diccionario.get('b')) #2
```

Métodos de Diccionarios

del(diccionario["cursos"])
diccionario.pop("edad")

setdefault()

Opera de dos formas. La primera como get()

```
diccionario = {'a':1, 'b':2, 'c':3, 'd':4}
print(diccionario.setdefault('b')) # 2
```

Agrega un elemento nuevo al diccionario.

```
diccionario.setdefault('e',5)
print(diccionario)
```

Métodos de Diccionarios

update()

Recibe como parámetro otro diccionario. Si tienen claves iguales, actualiza el valor de la clave; si no si tiene la clave, el par clave-valor es agregado al diccionario.

```
dicc1 = {'a':1, 'b':2, 'c':3, 'd':4}
dicc2 = {'c':6, 'b':7, 'e':8, 'f':12}
dicc1.update(dicc2)
print(dicc1) # {'a': 1, 'b': 7, 'c': 6, 'd': 4, 'e': 8, 'f': 12}
```

User Input

The Input Function

- The **input()** function provides a way to get information from the user.

```
>>> firstName = input("What is your first name? ")
What is your first name? Bob
>>> print("Hello " + firstName +"!")
Hello Bob!
>>>
```

Métodos para Strings

strip(), rstrip() y lstrip()

Elimina los espacios (catacteres) iniciales y finales de una cadena.

```
data = '     mi texto mi texto     '
data2 = '#'
print(data2+data+data2)
print('Texto sin espacios: ')
print(data2+data.strip()+data2)
```

```
print(len(data))
print(len(data.strip()))
```

Métodos para Strings

lower() y upper()

Convierte toda una cadena a mayúsculas o minúsculas.

```
data = 'mi texto'
data2 = 'MI TEXTO'
print(data.upper())
print(data2.lower())
```

Métodos para Strings

capitalize() y swapcase()

Solo la primera letra es mayúscula.
Invierte mayúsculas por minúsculas y viceversa.

```
data = 'mi texto mas extenso'
data2 = 'Mi Texto Mas Extenso X2'
print(data.capitalize())
print(data2.swapcase())
```

Métodos para Strings

mitexto[inicio:fin:paso]

Para copiar parte de un string.

```
str1 = "123456789*123456789*123456789*123456789*123456789*"
print("texto original: ", str1)
str2 = str1[0:15]
print("Substring 0 to 15: ", str2)   #
str2 = str1[10:]
print("Substring 10+: ", str2)   #
str2 = str1[0::2]
print("Substring 2 by 2: ", str2)   #
```

Métodos para Strings

split()
split(separador, max_separación)

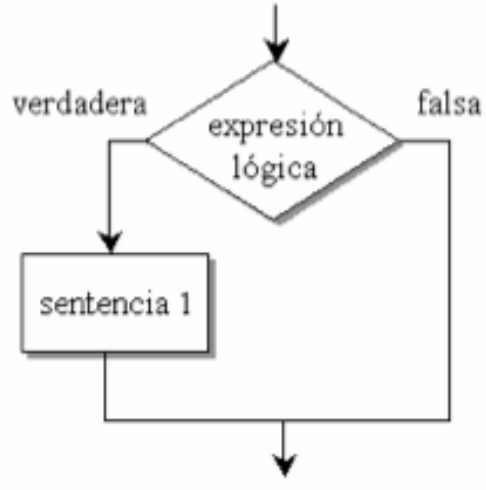
Divide un string en subcadenas.

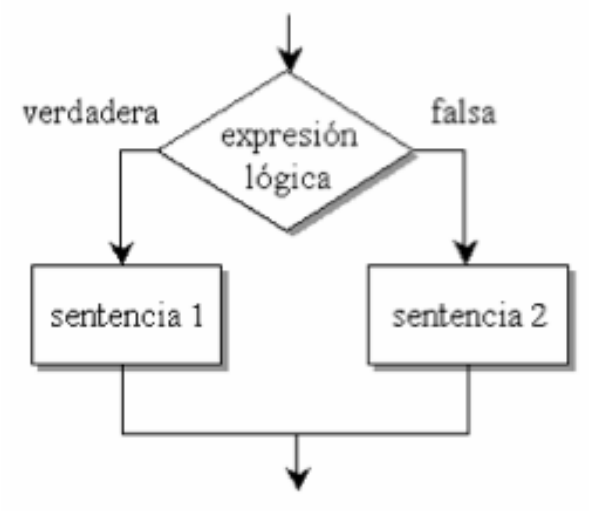
```
txt = "welcome to the jungle"
x = txt.split()
print(x)
```

```
txt = "apple#banana#cherry#orange"
# setting the max parameter to 1, will return a list with 2 elements!
x = txt.split("#", 1)
print(x)
```

If Functions and Loops

Condicional IF





If/Else Function

- Open a blank script and save it as 04_if-vlan.py.
- Create a simple **if** function that compares two values and prints the results.
- Run the script and troubleshoot any errors.
- Change the values to test the **else** print statement.

```
nativeVLAN = 1
dataVLAN = 100
if nativeVLAN == dataVLAN:
    print("The native VLAN and the data VLAN
are the same.")
else:
    print("This native VLAN and the data VLAN
are different.")
```

```
if 'la' in 'hola':
    print '¡Está!'
```

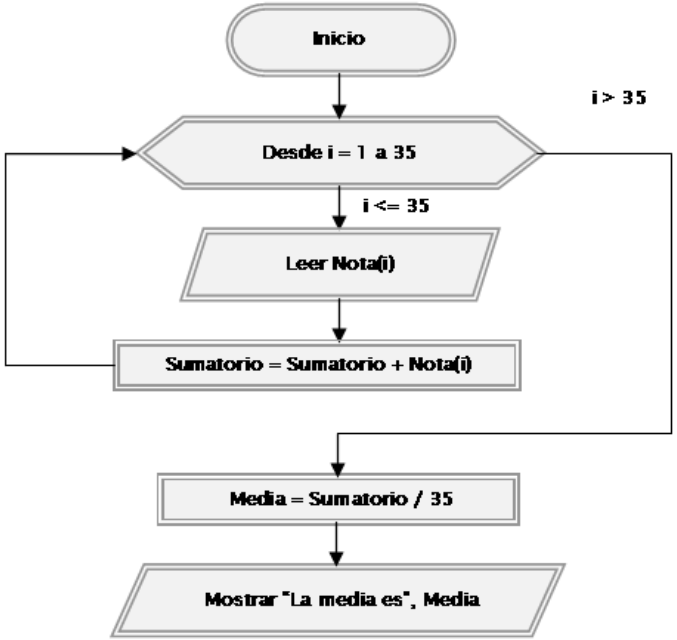
```
nativeVLAN = 100
dataVLAN = 100
if nativeVLAN == dataVLAN or dataVLAN < 100:
    print("Éxito  XD  ")
else:
    print("Falla :'(  ")
```

If/Elif/Else Function

- Open a blank script and save it as 05_if-acl.py.
- Create a more complex **if** function that takes user input and includes an **elif** loop.
- Note that the input needs to be converted to an integer.

```
aclNum = int(input("What is the IPv4 ACL
number? "))
if aclNum >= 1 and aclNum <= 99:
    print("This is a standard IPv4 ACL.")
elif aclNum >=100 and aclNum <= 199:
    print("This is a extended IPv4 ACL.")
else:
    print("This is not a standard or extended
IPv4 ACL.")
```

Ciclo For



For Loop

- A for loop iterates through items in a list, dictionary, or other sequenced data type.
- The variable name “item” is arbitrary and can be anything the programmer chooses.

```
>>> devices=["R1","R2","R3","S1","S2"]
>>> for item in devices:
        print(item)


R1
R2
R3
S1
S2
>>>
```

For Loop with Embedded If

- Using an If loop inside the For loop

```
>>> devices=["R1","R2","R3","S1","S2"]
>>> for item in devices:
        if "R" in item:
                print(item)


R1
R2
R3
>>>
```

Use a For Loop to Create a New List

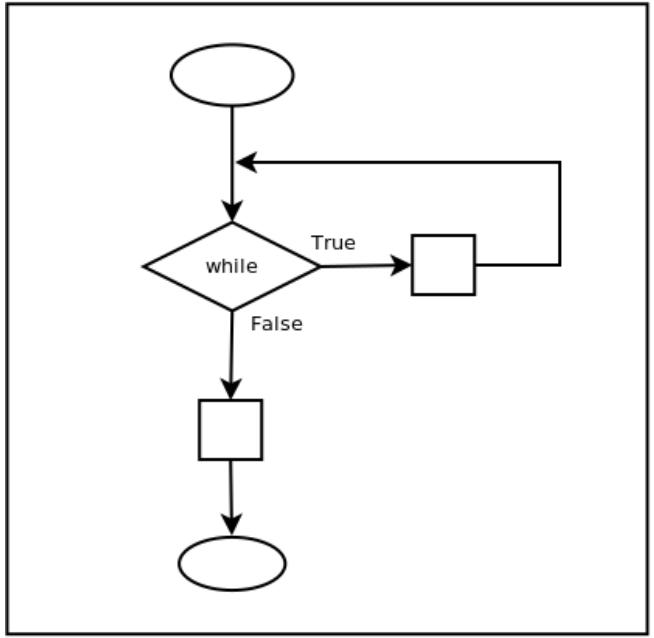
- Create an empty list called switches.
- Iterate through the devices list to create the switch list.

```
>>> devices=["R1","R2","R3","S1","S2"]
>>> switches=[]
>>> for item in devices:
        if "S" in item:
                switches.append(item)

>>> print(switches)
['S1', 'S2']
>>>
```

```
x = range(5,50,2)
for n in x:
  print(n,end="-")
```

While



Create a While Loop

- Open a blank script and save it as 06_while-loop.py.
- Create a program with a while loop that counts to a user's supplied number.
  - Convert the string to an integer: **x = int(x)**
  - Set a variable to start the count: **y = 1**
  - **While y <= x**, print the value of y and increment y by 1.

```
x=input("Enter a number to count to: ")
x=int(x)
y=1
while y<=x:
    print(y)
    y=y+1
```

Modify the While Loop to Use Break

- Modify the while loop to use a Boolean check and break to stop the loop.
  - Replace **while y<=x** with **while True**
  - Add an if function to break the loop when **y>x**.

```
x=input("Enter a number to count to: ")
x=int(x)
y=1
while True:
    print(y)
    y=y+1
    if y>x:
        break
```

Use a While Loop to Check for User Quit

- Add another while loop to the beginning of the script which will check for a quit command.
- Add an if function to the while loop to check for ‘q’ or ‘quit’ .

```
while True:
    x=input("Enter a number to count to: ")
    if x == 'q' or x == 'quit':
        break
x=int(x)
y=1
while True:
    print(y)
    y=y+1
    if y>x:
        break
```

File Access

Files in Python

To work with files in Python, use the open() function. It takes two parameters; filename, and mode. There are four different methods (modes) for opening a file:

Mode	Description
r	Read - Default value. Opens a file for reading, error if the file does not exist
a	Append - Opens a file for appending, creates the file if it does not exist
w	Write - Opens a file for writing, creates the file if it does not exist
x	Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

t	Text - Default value. Text mode
b	Binary - Binary mode (e.g. images)

Read Files

To open a file for reading…

f = open("demofile.txt") ## is equal to f = open("demofile.txt", "rt")

Is a good practice to close the file.

```
### demofile.txt
a. Hello World!
b. This file is a text file
c. This file is for testing.
d. Well done!
e. Last chance
f. Bye!
```

```
miLista=[]
f = open("demofile.txt", "r")
for x in f:
  print(x)
  print(x.lower())
  miLista.append(x.upper())
f.close()
print(miLista)
```

Read Files

To open a file for reading…

f = open("demofile.txt") ## is equal to f = open("demofile.txt", "rt")

```
### demofile.txt
a. Hello World!
b. This file is a text file
c. This file is for testing.
d. Well done!
e. Last chance
f. Bye!
```

```
f=open("demofile.txt","r")
print(f.read())
f.close()

f = open("C:\\Users\isan\demofile.txt", "r")
print(f.readline())
print(f.readline())
print(f.read())
f.close()
```

Write Files

To write in a file, you must add the correct parameter to the open() function:

x - Create - will create a file, returns an error if the file exist

a - Append – Will create if not exists the file or append if exists

w - Write – Will create if not exists the file or overwrite if exists

demofile.txt
a. Hello World!
b. This file is a text file
c. This file is for testing.
d. Well done!
e. Last chance
f. Bye!

```
f = open("demofile.txt", "a")
f.write("\nThe file is bigger \n and more")
f.close()
```

```
f = open("demofile.txt", "w")
f.write("Bye old lines!")
f.close()
```

```
f = open("demofile.txt", "r")
print(f.read())
```

Write Files

To write in a file, you must add the correct parameter to the open() function:

x - Create - will create a file, returns an error if the file exist

a - Append – Will create if not exists the file or append if exists

w - Write – Will create if not exists the file or overwrite if exists

```
f = open("newfile.txt", "x")
f.close()
print("File Created")

f = open("newfile.txt", "x") ##Will fail
```

Delete Files

Example to delete a file:

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
    print("The file was removed")
else:
    print("The file does not exist")
##deldir is an empty directory
os.rmdir("deldir")
```

CISCO